Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Eva Križman

Segmentacija slik z uporabo največjega pretoka

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentorica: izr. prof. dr. Arjana Žitnik

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Segmentacija slik z uporabo največjega pretoka

Tematika naloge:

Problem največjega pretoka je eden od klasičnih problemov kombinatorične optimizacije, uporaben pri reševanju mnogih problemov iz prakse. Eden od njih je segmentacija slik. Problem največjega pretoka oziroma njemu dualen problem najmanjšega prereza lahko uspešno uporabimo pri segmentaciji slik za ločevanje med ozadjem in objekti v ospredju slike.

V diplomskem delu predstavite problem največjega pretoka, problem najmanjšega prereza in povezavo med njima. Predstavite tudi nekaj algoritmov za njuno reševanje. Na kratko razložite pojem segmentacije slik. Nato podrobno predstavite postopek segmentacije s pomočjo največjega pretoka.

Posebna zahvala gre mentorici izr. prof. dr. Arjani Žitnik, za vso strokovnost in pomoč pri nastajanju diplomske naloge. Zahvala gre tudi mojemu fantu Mateju in prijateljem, ki so me prenašali in pomagali v vseh letih študija. Nazadnje bi se rada zahvalila svoji družini, mami Anici in očetu Marjanu za vso podporo in potrpežljivost.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
BFS	Breadth First Search	iskanje v širino
DFS	Depth First Search	iskanje v globino
CM	Clustering Method	razvrščanje v skupine
KMC	k - Means Clustering	metoda k-tih voditeljev
TM	Tresholding Method	pragovna metoda

Povzetek

Naslov: Segmentacija slik z uporabo največjega pretoka

Avtor: Eva Križman

V diplomski nalogi obravnavamo segmentacijo slik s pomočjo največjega pretoka. V prvem delu podrobneje predstavimo problem največjega pretoka in njemu dualen problem najmanjšega prereza. Predstavimo tudi dva alogritma za reševanje teh dveh problemov, to sta algoritem Forda in Fulkersona in Diničev algoritem.

V drugem delu diplomske naloge predstavimo pojem segmentacije slik in naštejemo nekaj metod, s katerimi se segmentacija izvaja. Nekatere od njih tudi opišemo. To so pragovna metoda, metoda razvrščanja v skupine, metoda regij in segmentacija na grafih. V zadnjem poglavju si podrobno pogledamo segmentacijo na grafih. Sliko predstavimo z uteženim grafom in na temu grafu poiščemo največji pretok oziroma najmanjši prerez. S pomočjo prereza potem ločimo slikovne pike na tiste, ki pripadajo ospredju in tiste, ki pripadajo ozadju. Vse algoritme predstavimo s psevdokodo in analiziramo njihovo časovno zahtevnost. Na koncu predstavimo tudi probleme, ki se pojavljajo pri tovrstni segmentaciji.

Ključne besede: graf, omrežje, največji pretok, najmanjši prerez, algoritem Forda in Fulkersona, segmentacija slik.

Abstract

Title: Image segmentation using maximum flow

Author: Eva Križman

In this thesis we consider image segmentation using maximum flow. In the first part of the thesis we present in detail the maximum flow problem and its dual problem, the minimum cut problem. We describe two algorithms for solving these two problems, the Ford-Fulkerson algorithm and Dinic algorithm.

In the second part of the thesis we introduce the concept of image segmentation and we list some methods that are used for image segmentation. We describe some of them: the thresholding method, clustering methods, the region-growing methods and segmentation on graphs. In the last chapter we present in more detail segmentation on graphs using maximum flow. We represent a given image with a weighted graph. Then we find maximum flow and minimum cut in this graph. Using minimum cut we can separate pixels to those that belong to the foreground and those that belong to the background. Pseudo code is given for all the algorithms used and their time complexity is analyzed. At the end of the thesis we discuss some problems that arise in this kind of segmentation.

Keywords: graph, network, maximum flow, minimum cut, Ford–Fulkerson algorithm, image segmentation.

Poglavje 1

Uvod

Predstavljajmo si skupino podjetnikov, ki želijo z letalom iz konference v mestu A priti domov v mesto B. Žal so vsi direktni leti že zasedeni, zato so primorani leteti s prestopanjem. Kapacitete na povezavah med posameznimi mesti so prosta mesta na letalu. Želimo najti tako razporeditev, da bomo iz mesta A do mesta B še isti dan pripeljali čim več podjetnikov. Takemu problemu pravimo problem največjega pretoka.

Soroden problem je imela ameriška vojska v času hladne vojne, ko jo je zanimalo, koliko cestnih povezav morajo uničiti, da bodo preprečili prihod sovjetske vojske v vzhodno Evropo. To je problem najmanjšega prereza, ki je povezan s problemom največjega pretoka. Na ukaz ameriške vojske sta se T. E. Harris in F. S. Ross začela ukvarjati s tem problemom in ga tudi prvič uradno objavila, glej [17]. Sicer nista našla metode, ki bi zagotovila optimalno rešitev, sta pa s tem navdušila Forda in Fulkersona, da sta začela raziskovati ta problem. Tako sta izpeljala Ford–Fulkersonov algoritem, ki najde največji pretok in najmanjši prerez v danem omrežju in zagotavlja optimalno rešitev. Osnovni algoritem ima lahko eksponentno število korakov glede na velikost omrežja. Skozi desetletja so številni raziskovalci izboljšali časovno zahtevnost Ford–Fulkersonovega algoritma [1].

Leta 1989 so prvič uporabili metodo najmanjšega prereza v računalniškem vidu. Pokazali so, da je najmanjši prerez uporaben pri obnavljanju binarnih

slik. V poznih 90-tih so z novimi tehnikami računalniškega vida najmanjši prerez začeli uporabljati na nebinarnih slikah, glej [4].

Obdelava slik je bila razvita zaradi treh glavnih problemov. Prvi problem je bila digitalizacija slike in njen enostavnejši zapis za lažje tiskanje, prenos in shranjevanje. Drugi problem je restavriranje slike in lažje razumevanje določenih segmentov na sliki. Zadnji problem je zgodnja faza računalniškega vida, ki uporablja segmentacijo slik [15].

Mi se bomo osredotočili na problem segmentacije slik. Segmentacija vsaki slikovni piki dodeli oznako. Slikovne pike z isto oznako tvorijo določen objekt na sliki. Rezultat segmentacije slik je nova slika, ki je lažja za analizo in nadaljno obdelavo. Orodja, pri katerih se srečujemo s segmentacijo, so na primer Slikar, Photoshop in Gimp, v katerih lahko spreminjamo označene dele na sliki. Pomembna uporaba segmentacije je v medicini. Zdravniki lahko z njeno pomočjo enostavno odkrijejo tumor, nepravilnosti ali si pomagajo pri razumevanju človeškega telesa (lahko označijo samo kosti in s slike bodo odtranjeni vsi ostali segmenti, ki niso kosti), glej [21, 10].

Opišimo na kratko zgradbo diplomskega dela. Uvodnemu poglavju sledi poglavje o grafih. Definiramo grafe in nekatere njihove lastnosti. Opišemo algoritem iskanja v širino, s katerim preiščemo graf v linearnem času. Graf z dodatno informacijo o kapacitetah povezav imenujemo *omrežje*. V drugem poglavju definiramo problem največjega pretoka in najmanjšega prereza na omrežju in opišemo pomembnejše lastnosti ter povezavo med njima. Za reševanje teh problemov opišemo algoritem Forda in Fulkersona in Diničev algoritem. V tretjem poglavju definiramo segmentacjo slik in naštejemo nekaj metod, ki se uporabljajo v praksi. Nekatere od njih podrobneje predstavimo: pragovna metoda, metoda k-tih voditeljev, metoda regij in segmentacija na grafih s pretokom. Za vsako metodo v programu MATLAB naredimo primer slike in pokažemo kakšen, rezultat segmentacije lahko pričakujemo. V zadnjem poglavju si podrobno pogledamo segmentacijo na grafih, kjer segmentacijo slik prevedemo na problem najmanjšega prereza. Poglobimo se v samo reševanje problema segmentacije na grafih. Sliko predstavimo z uteženim

grafom in na njem poiščemo največji pretok oziroma najmanjši prerez. S pomočjo prereza razdelimo slikovne pike na tiste, ki pripadajo ospredju in na tiste, ki pripadajo ozadju. Problem segmentacije na grafih predstavimo v algoritmih in zanje izpeljemo tudi časovno zahtevnost. Na koncu opišemo še probleme, ki se pojavljajo pri tovrstni segmentaciji.

Poglavje 2

Osnovno o grafih

V tem poglavju bomo definirali osnovne pojme s področja teorije grafov. Navezovali se bomo na vire [22, 3].

Definicija 2.1. *Graf* je urejen par $G = (V, E)$, pri čemer je V končna neprazna množica *vozlišč* grafa G in množica $E \subseteq \binom{V}{2}$, predstavlja podmnožico *povezav* grafa G.

Definicija 2.2. Če je $\{u, v\}$ povezava v grafu G, potem pravimo, da sta vozlišči u in v *sosednji*. Povezava $e = \{u, v\}$ ima *krajišči* u in v.

Povezavo $\{u, v\}$ bomo krajše zapisovali uv ali vu. Z $V(G)$ bomo označevali množico vozlišč, z $E(G)$ pa množico povezav grafa G.

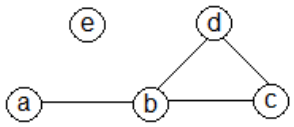
Da sta vozlišči u, v povezani s povezavo e, lahko zapišemo z $uv \in E(G)$ ali $u \sim_G v$. Če vemo za kateri graf gre, zapišemo krajše $uv \in E$ ali u $\sim v$. Število vozlišč grafa G označimo z $n = |V(G)|$, z $m = |E(G)|$ označujemo število povezav grafa G.

Definicija 2.3. *Stopnja vozlišča* v je število povezav grafa G, ki imajo vozlišče v za svoje krajišče. Vozliščem stopnje 0 pravimo *izolirana* vozlišča, vozliščem stopnje 1 pa *listi*.

Stopnjo vozlišča v v grafu G označimo z $\deg(v)$. Stopnjo vozlišča, ki ima najmanjšo stopnjo v grafu G, označimo z $\delta(G)$, največjo stopnjo vozlišča pa z $\Delta(G)$. Torej za vsako vozlišče $v \in V(G)$ velja:

$$\delta(G) \leq \deg(v) \leq \Delta(G).$$

Za zgled si poglejmo naslednji graf $G = (\{a, b, c, d, e, \}, \{ab, bc, bd, cd\})$. Na sliki 2.1 lahko opazimo, da so stopnje vozlišč enake $\deg(a) = 1$, $\deg(b) = 3$, $\deg(c) = \deg(d) = 2$ in $\deg(e) = 0$. Vidimo, da je e izolirano vozlišče in vozlišče a list.



Slika 2.1: Graf $G = (\{a, b, c, d, e\}, \{ab, bc, bd, cd\})$.

Definicija 2.4. Naj bosta vozlišči u in v sosednji v grafu $G = (V, E)$. Grafu G lahko povezavo uv *odstranimo* in dobimo graf $G - uv$, za katerega velja

$$V(G - uv) = V(G) \text{ in } E(G - uv) = E(G) \setminus \{uv\}.$$

Če vozlišči u in v nista sosednji, potem lahko v grafu G povezavo uv *dodamo*. Novi graf označimo z $G + uv$ in zanj velja

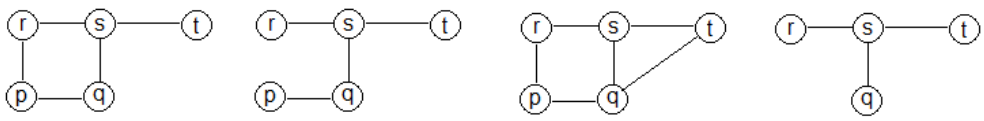
$$V(G + uv) = V(G) \text{ in } E(G + uv) = E(G) \cup \{uv\}.$$

Definicija 2.5. Naj bo v vozlišče v grafu $G = (V, E)$. Vozlišče v lahko iz grafa G odstranimo in dobimo graf $G - v$. Zanj velja

$$V(G - v) = V(G) \setminus \{v\} \text{ in } E(G - v) = E(G) \setminus \{uv, uv \in E(G)\}.$$

Definicija 2.6. Graf $G_1 = (V_1, E_1)$ je *podgraf* grafa $G = (V, E)$, če velja $V_1 \subseteq V$ in $E_1 \subseteq E$. Z drugimi besedami, če iz grafa G zaporedno odstranjujemo vozlišča ali povezave, lahko dobimo podgraf G_1.

Na sliki 2.2 imamo na levi osnovni graf G. Na naslednji sliki grafu G odstranimo povezavo rp. Na predzadnji sliki grafu G dodamo povezavo qt. Na sliki desno je odstranjeno vozlišče p iz grafa G.



Slika 2.2: Graf G, graf $G - rp$, graf $G + qt$ in graf $G - p$.

V grafu vsaki povezavi e ustreza par *usmerjenih povezav* (u, v) in (v, u). Vozlišče u je *začetek*, vozlišče v pa *konec usmerjene* povezave (u, v). Usmerjena povezava se v grafih največkrat označuje s puščico. Povezavo $e = (u, v)$, ki je usmerjena od u proti v, označimo krajše $e = uv$.

Definicija 2.7. *Usmerjen graf* ali *digraf* je urejen par množic grafa $G = (V(G), E(G))$, kjer je $V(G)$ poljubna množica vozlišč, $E(G)$ pa množica usmerjenih povezav.

Definicija 2.8. Naj bo v vozlišče v digrafu G. Množico vseh povezav, ki se začnejo v vozlišču v, označimo z $\mathrm{Out}(v)$. Velja:

$$\mathrm{Out}(v) = \{e \in E(G), \text{začetek}(e) = v\}.$$

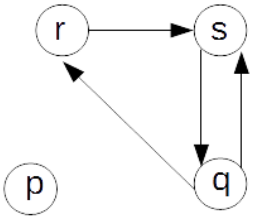
Množico vseh povezav, ki se končajo v vozlišču v, označimo z $\mathrm{In}(v)$ in velja:

$$\mathrm{In}(v) = \{e \in E(G), \text{konec}(e) = v\}.$$

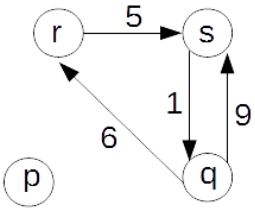
Na sliki 2.3 je usmerjen graf $G = (V, E)$ z množico vozlišč $V = \{p, q, r, s\}$ in množico povezav $E = \{rs, sq, qs, qr\}$. Samo ena povezava se začne v vozlišču r, to je povezava rs. Zato je $\mathrm{Out}(r) = \{rs\}$. Samo ena povezava se konča v vozlišču r, to je povezava qr. Zato je $\mathrm{In}(r) = \{qr\}$.

Definicija 2.9. *Utežen graf* je graf $G = (V(G), E(G))$ z dano preslikavo $c : E(G) \to \mathbb{R}$, ki vsaki povezavi grafa G priredi utež (ceno).

Na sliki 2.4 je utežen usmerjen graf G. Velja $c(rs) = 5$, $c(sq) = 1$, $c(qs) = 9$, $c(qr) = 6$.



Slika 2.3: Usmerjen graf G.



Slika 2.4: Usmerjen graf G z utežmi na povezavah.

Definicija 2.10. Naj bo G graf ali digraf. Potem je zaporedje

$$S = v_0, v_1, v_2, ..., v_{k-1}, v_k$$

sprehod, če velja $v_i v_{i+1} \in E(G)$ za $i = 0, ..., k-1$.

Vozlišči v_0 in v_k imenujemo *krajišči* sprehoda S. Sprehod lahko isto povezavo obišče večkrat, prav tako lahko posamezno povezavo uporabi v različnih smereh. Število uporabljenih povezav v sprehodu označimo s $|S|$ in mu pravimo *dolžina* sprehoda S. Kadar sprehod nobene povezave ne obišče več kot enkrat, pravimo, da gre za *enostaven sprehod*. Če pa ima sprehod S paroma različna vozlišča, torej za vsak $i, j \in \{0, ...k\}$ velja $v_i \neq v_j$, potem takemu sprehodu pravimo *pot*.

Definicija 2.11. *Obhod* je sprehod, pri katerem sovpadata začetna in končna točka.

Definicija 2.12. *Cikel* je sprehod

$$w_0, w_1, ..., w_{k-1}, w_k,$$

za katerega velja $k \geq 3$ in $w_0 = w_k$, ostala vozlišča pa so paroma različna.

Iz predhodnih definicij opazimo, da so poti in cikli enostavni sprehodi.

Definicija 2.13. *Kvazi pot* v digrafu $G = (V, E)$ je zaporedje

$$v_0, e_1, v_1, e_2, ..., e_k, v_k,$$

kjer $v_i \in V$ in $e_i \in E$ ter velja $e_i = v_{i-1}v_i$ ali $e_i = v_iv_{i-1}$. Zahtevamo, da se nobena povezava v zaporedju ne ponovi.

Definicija 2.14. Graf $G' = (V', E')$ je *vpet podgraf* grafa $G = (V, E)$, če velja $G' \subseteq G$ in $V(G') = V(G)$. Vpete podgrafe dobimo zgolj z odstranjevanjem povezav grafa.

Definicija 2.15. *Drevo* je povezan graf brez ciklov.

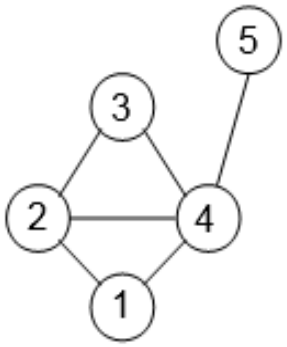
Pravimo, da je T *vpeto drevo* v grafu G, če je T drevo in hrati vpet podgraf grafa G.

Definicija 2.16. Naj bo G graf z n vozlišči, označenimi z $1, 2, 3, ..., n$. *Matrika sosednosti* $M(G)$ grafa G je matrika razsežnosti $n \times n$, v kateri element v j-tem stolpcu i-te vrstice pove število povezav, ki povezujejo vozlišči i in j.

Matrika sosednosti $M(G)$:

$$M = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} \end{bmatrix}, \; x_{ij} = \begin{cases} 1, & \text{če } x_i \sim x_j, \\ 0, & \text{sicer.} \end{cases}$$

Na sliki 2.5 levo imamo prikazan graf G. Vozlišče 1 je povezano z vozliščema 2 in 4, zato imamo v matriki sosednosti v prvi vrstici enico na



$$M = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Slika 2.5: Graf G in njegova matrika sosednosti.

drugem in četrtem mestu. Vozlišča niso povezana sama s seboj, zato imamo po diagonali ničle. Matrika sosednosti, ki pripada grafu G, je na sliki 2.5 desno.

Preiskovanje imenujemo pregledovanje drevesa ali grafa z določenim algoritmom, dokler ne najdemo iskanega vozlišča ali pregledamo celotnega grafa. *Iskanje v širino* ali Breadth First Search (BFS) spada med enostavnejše algoritme preiskovanja. Imamo graf $G = (V, E)$. Iskanje/preiskovanje začnemo z začetnim vozliščem $v \in V$. Vozlišča, ki so sosednji vozlišču v, lahko takoj označimo kot kandidate za obisk. Kandidate obiskujemo po vrsti in jih sproti označujemo kot obiskane. Postopek ponavljamo, dokler ne obiščemo vseh vozlišč oziroma ne najdemo iskano vozlišče. Psevdokoda za iskanje v širino je predstavljena v algoritmu 1.

Ocenimo časovno zahtevnost algoritma 1. Podatkovno vrsto ustvarimo v konstantnem času, neodvisno od vrednosti preostalih spremenljivk, medtem ko označevanje vozlišč kot neobiskana vzame $O(|V|)$ časa. Dodajanje in označevanje vozlišč zahteva konstantni čas. V zanki while pregledamo vsako povezavo največ dvakrat, za kar porabimo skupno $O(|E|)$ časa. Če seštejemo časovne zahtevnosti posameznih delov, dobimo časovno zahtevnost $O(|V| + |E|)$.

Algoritem 1: Iskanje v širino (BFS)

Input: Povezan graf G, začetno vozlišče s

Output: Vsa vozlišča so označena kot obiskana

begin
  Ustvarimo novo podatkovno vrsto Q;
  foreach $v \in V$ **do**
    vsa vozlišča v označi kot ne obiskana;
  end
  V vrsto Q dodaj s;
  Vozlišče s označi kot obiskano;
  while Q ni prazen **do**
    x dobi vrednost prvega vozlišča iz vrste Q;
    for vsak y, ki je sosed x-a **do**
      if y ni obiskan **then**
        v vrsto Q dodaj vozlišče y;
        vozlišče y označi kot obskano;
      end
    end
  end
end

Iskanje v širino je osnova za reševanje naslednjih problemov v teoriji grafov:

- preverjanje dvodelnosti grafa,
- iskanje najkrajše poti v grafu,
- iskanje vseh vozlišč v nekem povezanem podgrafu.

Algoritem 2: Najkrajša pot

Input: Povezan graf G, začetno vozlišče s, iskano vozlišče t

Output: Vrne najkrajšo pot od vozlišča s do iskanega vozlišča t

```
begin
    Ustvarimo novo podatkovno vrsto Q;
    foreach v ∈ V(G) \ {s} do
        vsa vozlišča v označi kot neobiskana;
        vsem vozliščem nastavi pot[v] na NULL;
    end
    V vrsto Q dodaj s; //s je začetno vozlišče
    Vozlišče s označi kot obiskano;
    while Q ni prazen do
        x dobi vrednost prvega vozlišča iz vrste Q;
        for vsak y, ki je sosed x do
            if y ni obiskan then
                vozlišče y označi kot obiskano;
                v pot[y] nastavi predhodnika x;
                if y je iskano vozlišče t then
                    vrni pot[y];
                end
                v vrsto Q dodaj vozlišče y;
            end
        end
        Iz Q odstrani prvo vozlišče.
    end
end
```

Za nas bo predvsem pomemben algoritem za iskanje najkrajše poti v grafu. Algoritem 1 bomo nekoliko dopolnili, da nam bo vrnil najkrajšo pot do iskanega vozlišča. Psevdokoda je predstavljena v algoritmu 2. Tako kot v algoritmu 1, tudi tukaj najprej ustvarimo podatkovno vrsto in označimo vsa vozlišča kot neobiskana. Inicializiramo tabelo pot. Vsako vozlišče kaže na

svojega predhodnika, razen začetnega vozlišča. Torej je v tabeli pot na i-tem mestu predhodnik vozlišča i na poti. Do sedaj smo porabili ponovno $O(|V|)$ časa. Preden vstopimo v zanko while, začetno vozlišče dodamo v vrsto in ga označimo kot obiskanega. To ne doprinese ničesar k časovni zahtevnosti, saj vse opravimo v času $O(1)$. Vstopimo v zanko while in pogledamo prvo vozlišče, ki se nahaja v vrsti. V zanki se sprehodimo do vseh sosedov in preverimo ali smo že našli iskano vozlišče. Če smo našli iskano vozlišče, vrnemo pot in končamo z algoritmom. Če ga še nismo našli, nadaljujemo z iskanjem. Časovna zahtevnost algoritma je podobna kot prej $O(|V| + |E|)$, saj vsako povezavo obiščemo največ dvakrat.

Poglavje 3

Problem največjega pretoka

Največji pretok in najmanjši prerez sta komplementarna problema. Predstavljajmo si cestno omrežje, ki je predstavljeno z usmerjenim grafom. V grafu mesta predstavljajo vozlišča, ceste pa povezave med njimi. Začetno mesto imenujemo izvor in ciljno mesto ponor. Število avtomobilov, ki se lahko v določenem času pripeljejo po dani cesti, je kapaciteta te ceste. Kapaciteta ne sme biti negativna. Problem je najti največje število avtomobilov, ki lahko potujejo od začetnega do ciljnega mesta. Temu problemu pravimo problem največjega pretoka. Problem najmanjšega prereza je določitev povezav z najmanjšo vsoto kapacitet, ki jih lahko odstranimo in s tem preprečimo povezavo od izvora do ponora. Zanimivo je to, da je problem največjega pretoka tesno povezan s problemom iskanja najmanjšega prereza. L. R. Ford, Jr. in D. R. Fulkerson sta dokazala, da sta si ta dva problema dualna v smislu, da je vrednost največjega pretoka enaka vrednosti najmanjšega prereza [7]. V tem poglavju bomo spoznali osnovne lastnosti problema največjega pretoka in najmanjšega prereza. Za iskanje največjega pretoka bomo predstavili algoritem Forda in Fulkersona ter Diničev algoritem. Pri opisu algoritmov smo si pomagali z viri [23, 11, 17, 16].

3.1 Največji pretok in najmanjši prerez

Dan imamo usmerjen graf $G = (V, E)$. V grafu imamo vozlišči $s, t \in V$, ki predstavljata *izvor* in *ponor*. Na množici povezav imamo definirano preslikavo $c : E \to \mathbb{R}_0^+$. Vrednost $c(i, j)$ imenujemo *kapaciteta* povezave (i, j), ki jo bomo krajše označevali s c_{ij}. Četverko $N = (G, s, t, c)$ imenujemo *omrežje*. Z E_N bomo označevali vse povezave v omrežju N, z V_N bomo označevali vsa vozlišča v omrežju N. V nadaljevanju bomo predpostavili, da imamo dano omrežje $N = (G, s, t, c)$

Definicija 3.1. Preslikava $f : E_N \to \mathbb{R}^+$ je *pretok* v omrežju N, če velja:

1. $f(e) \leq c(e)$, za vsako povezavo e v omrežju N,
2. $\sum_{e \in \mathrm{In}(v)} f(e) = \sum_{e \in \mathrm{Out}(v)} f(e)$ za vsako vozlišče $v \in V_N \setminus \{s, t\}$.

Definicija 3.2. *Velikost pretoka* f v omrežju N označimo z $|f|$ in ga definiramo kot:

$$|f| = \sum_{e \in \mathrm{Out}(s)} f(e) - \sum_{e \in \mathrm{In}(s)} f(e).$$

Maksimalni pretok f^* je pretok, ki ima v omrežju N največjo velikost, torej zanj velja

$$|f| \leq |f^*|, \text{ za vsak pretok } f \text{ v omrežju } N.$$

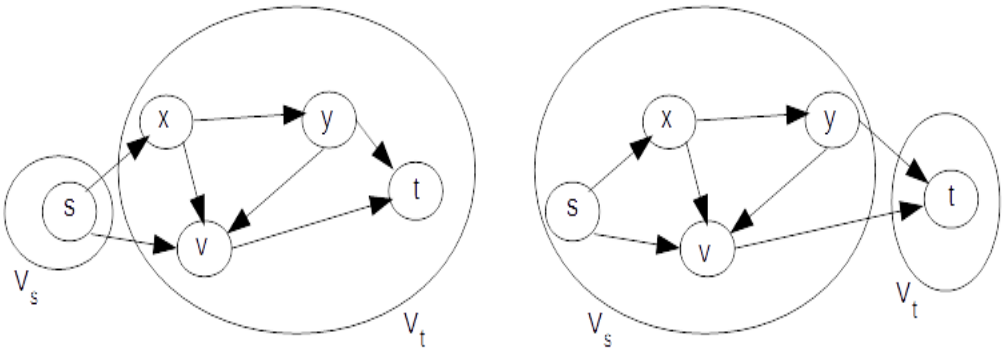
Zaradi lažje obravnave bomo preslikavo c razširili na množico $V \times V$ s predpisom $c_{uv} = 0$, če povezava $uv \notin E$ in enako tudi f razširimo na množico $V \times V$.

Definicija 3.3. Par množic V_s in V_t v grafu (V, E), za kateri velja:

$$V_s \bigcup V_t = V,$$
$$V_s \bigcap V_t = \emptyset,$$
$$s \in V_s \text{ in } t \in V_t$$

imenujemo *prerez* in ga označimo z $\langle V_s, V_t \rangle$.

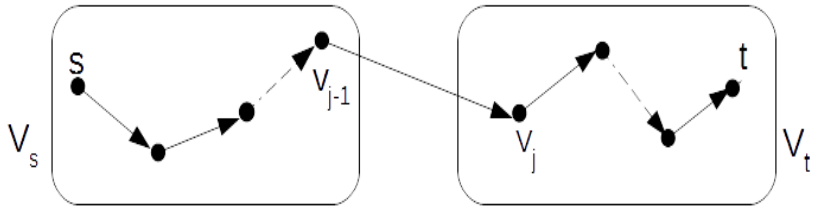
Na sliki 3.1 sta prikazana dva prereza istega omrežja. Prvi prerez je $\langle \{s\}, V - \{s\} \rangle = \langle \{s\}, \{x, y, v, t\} \rangle$, drugi pa $\langle V - \{t\}, \{t\} \rangle = \langle \{s, x, y, v\}, \{t\} \rangle$.



Slika 3.1: Različna prereza omrežja.

Izrek 3.1. *Naj bo $\langle V_s, V_t \rangle$ prerez omrežja N. Potem vsaka usmerjena $s - t$ pot v N vsebuje vsaj eno povezavo, ki povezuje vozlišče iz V_s z vozliščem iz V_t.*

Dokaz. Naj bo $P = s, v_1, v_2, ..., t$ usmerjena pot v omrežju N in naj bo v_j prvo vozlišče na poti P, ki je v množici V_t. Potem vozlišče v_{j-1} pripada množici V_s, povezava med njima je iskana povezava (slika 3.2). □



Slika 3.2: Usmerjena pot, pri kateri je povezava $v_{j-1}v_j$ v prerezu $\langle V_s, V_t \rangle$.

Definicija 3.4. *Kapaciteto prereza* $\langle V_s, V_t\rangle$ označimo kot $c\langle V_s, V_t\rangle$ in je vsota kapacitet povezav, ki povezujejo vozlišča iz V_s z vozlišči iz V_t:

$$c\langle V_s, V_t\rangle = \sum_{\substack{u\in V_s,\\ v\in V_t,}} c(uv).$$

Definicija 3.5. Če je $f : E_N \to \mathbb{R}$ pretok v omrežju N, je

$$f\langle V_s, V_t\rangle = \sum_{\substack{u\in V_s,\\ v\in V_t,}} f(uv) - \sum_{\substack{v\in V_t,\\ u\in V_s,}} f(vu)$$

tok skozi prerez $\langle V_s, V_t\rangle$.

Navedimo nekaj nekaj lastnosti pretokov. Naslednja trditev sledi iz prve lastnosti v definiciji 3.1.

Trditev 3.1. *Za vsak pretok* f *in vsak prerez* $\langle V_s, V_t\rangle$ *velja*

$$f\langle V_s, V_t\rangle \le c\langle V_s, V_t\rangle.$$

Trditev 3.2. *Naj bo* f *pretok in* $\langle V_s, V_t\rangle$ *prerez. Potem velja* $|f| = f\langle V_s, V_t\rangle$.

Dokaz. Zaradi lastnosti 2 iz definicije 3.1 velja:

$$\sum_{e\in \mathrm{Out}(v)} f(e) - \sum_{e\in \mathrm{In}(v)} f(e) = 0$$

za vsak $v \notin \{s, t\}$. Zato imamo

$$\begin{aligned}
|f| &= \sum_{e\in \mathrm{Out}(s)} f(e) - \sum_{e\in \mathrm{In}(s)} f(e) \\
&= \sum_{e\in \mathrm{Out}(s)} f(e) - \sum_{e\in \mathrm{In}(s)} f(e) + \sum_{v\in V_s\setminus\{s\}} \Big(\sum_{e\in \mathrm{Out}(v)} f(e) - \sum_{e\in \mathrm{In}(v)} f(e)\Big) \\
&= \sum_{v\in V_s} \Big(\sum_{e\in \mathrm{Out}(v)} f(e) - \sum_{e\in \mathrm{In}(v)} f(e)\Big) \\
&= \sum_{\substack{v\in V_s,\\ u\in V_t,}} f(vu) - \sum_{\substack{v\in V_t,\\ u\in V_s,}} f(vu) = f\langle V_s, V_t\rangle.
\end{aligned}$$

□

Definicija 3.6. *Minimalni prerez* omrežja N je prerez z najmanjšo kapaciteto.

Izrek 3.2. *Naj bo $\langle V_s, V_t\rangle$ prerez v omrežju N in naj bo f tak pretok, da velja*

$$f(e) = \begin{cases} c(e), & \text{če } e = uv \text{ in } u \in V_s,\ v \in V_t, \\ 0, & \text{če } e = uv \text{ in } u \in V_t,\ v \in V_s. \end{cases}$$

Potem je f največji pretok v omrežju N in $\langle V_s, V_t\rangle$ je najmanjši prerez.

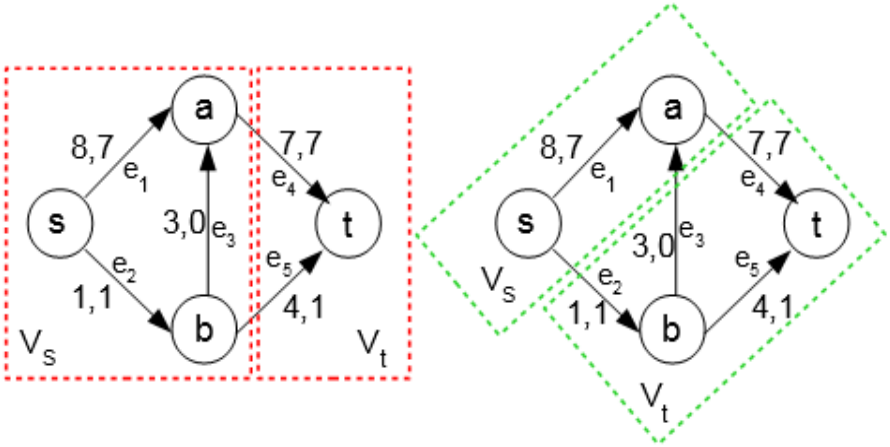
Dokaz. Naj bo f' poljuben pretok. Po trditvi 3.2 velja

$$|f'| = f'\langle V_s, V_t\rangle \leq c\langle V_s, V_t\rangle = f\langle V_s, V_t\rangle = |f|.$$

Torej je f največji pretok. Naj bo $\langle V_s', V_t'\rangle$ poljuben prerez. Potem po trditvi 3.2 velja

$$c\langle V_s, V_t\rangle = f\langle V_s, V_t\rangle = |f| = f\langle V_s', V_t'\rangle \leq c\langle V_s', V_t'\rangle.$$

Zadnja neenakost velja po trditvi 3.1. Torej je $\langle V_s, V_t\rangle$ najmanjši prerez. □



Slika 3.3: Pretok v omrežju. Na levi sliki je označen prerez, ki ni najmanjši. Na desni sliki je označen najmanjši prerez

Na sliki 3.3 imamo omrežje. Prva oznaka na povezavi pomeni kapaciteto, druga pa pretok. Na sliki levo imamo predstavljen prerez, ki gre skozi povezavi e_4 in e_5. Povezava e_4 je zasičena, saj zanjo velja $f(e_4) = c(e_4)$, medtem

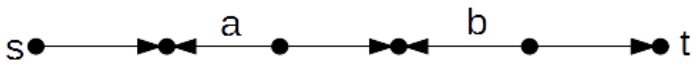
ko za povezavo e_5 to ne velja. Ker za povezavo e_5 velja $f(e_5) < c(e_5)$, povezava e_5 ni zasičena. Zato prerez na sliki levo ni nujno najmanjši prerez. Na sliki desno imamo tri povezave e_2, e_3, e_4, ki gredo skozi prerez omrežja. Za povezavi e_2 in e_4 velja $f(e) = c(e)$. Povezava e_3 je negativna povezava in zanjo velja $f(e_3) = 0$. Po izreku 3.2 smo našli najmanjši prerez omrežja.

Denimo, da je f pretok v omrežju N in obstaja kvazi pot Q:

$$Q = s, e_1, v_1, e_2, ..., e_k, t,$$

za katero velja $f(e_i) < c(e_i)$ za vsak $i = \{1, ..., k\}$. Povezavi e_i pravimo *pozitivna povezava*, če je usmerjena iz vozlišča v_{i-1} v v_i, če pa je povezava e_i usmerjena iz vozlišča v_i v v_{i-1} pravimo, da je e_i *negativna povezava*.

Pot je kvazi pot s samimi pozitivnimi povezavami. Sedaj bomo v poti dovolili tudi negativne povezave. Na sliki 3.4 sta negativni povezavi a in b.



Slika 3.4: Kvazi pot z dvema negativnima povezavama.

Definicija 3.7. Povezava $e \in E(G)$ je *zasičena* glede na pretok f, če velja $f(e) = c(e)$ in *nezasičena*, če velja $f(e) < c(e)$.

Odslej bomo namesto kvazi pot pisali krajše kar pot.

Definicija 3.8. Naj bo f pretok v omrežju N. *Nezasičena pot* je pot v omrežju N, za katero velja

$$f(e) < c(e), \text{ če je } e \text{ pozitivna povezava,}$$
$$f(e) > 0, \text{ če je } e \text{ negativna povezava.}$$

Na vsaki pozitivni povezavi lahko povečamo pretok, na negativni pa zmanjšamo.

Naj bo Q nezasičena pot. Potem za vsako povezavo e definiramo Δ_e, kot

$$\Delta_e = \begin{cases} c(e) - f(e), & \text{če je povezava pozitivna,} \\ f(e), & \text{če je povezava negativna.} \end{cases}$$

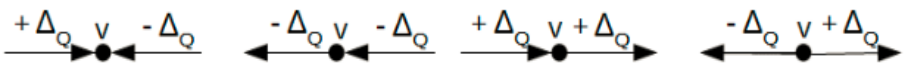
Vrednost Δ_e nam pove, za koliko še lahko povečamo tok na pozitivni povezavi in za koliko še lahko zmanjšamo tok na negativni povezavi. Z $\Delta_Q = \min_{e \in Q} \Delta_e$ bomo označili najmanjšo vrednost, za katero lahko spremenimo pretok na poti Q.

Izrek 3.3. *Naj bo f pretok v omrežju N in naj bo Q nezasičena pot z vrednostjo Δ_Q na povezavah. Definirajmo f^* kot*

$$f^* = \begin{cases} f(e) + \Delta_Q, & \text{če je povezava pozitivna,} \\ f(e) - \Delta_Q, & \text{če je povezava negativna,} \\ f(e), & \text{sicer.} \end{cases}$$

Potem je f^* tudi pretok in velja $|f^*| = |f| + \Delta_Q$.

Dokaz. Pokažimo najprej, da je f^* pretok. Po definiciji Δ_Q velja $0 \leq f^*(e) \leq c(e)$. Edina vozlišča, preko katerih se spremeni vrednost pretoka, so tista vozlišča, ki se nahajajo na nezasičeni poti Q. Pregledati moramo samo notranja vozlišča poti Q. Za vozlišče v imamo štiri različne možnosti za povečanje/zmanjšanje toka (slika 3.1). V vseh štirih primerih se pretok skozi vozlišče ohrani.



Slika 3.5: Štiri možnosti za vozlišče v znotraj poti Q.

Sedaj izračunajmo še velikost pretoka f^*. Naj bo e_1 prva povezava na poti Q. Če je e_1 pozitivna povezava, potem velja $f^*(e_1) = f(e_1) + \Delta_Q$, če pa je povezava e_1 negativna, potem zanjo velja $f^*(e_1) = f(e_1) - \Delta_Q$. Ker je s krajišče povezave e_1, velja

$$|f^*| = \sum_{e \in \mathrm{Out}(s)} f(e) - \sum_{e \in \mathrm{In}(s)} f(e) = |f| + \Delta_Q.$$

□

Izrek 3.4. *Naj bo f pretok v omrežju N. Potem je f največji pretok v omrežju N natanko tedaj, ko v omrežju N ne obstaja nezasičena pot.*

Dokaz. ($\Rightarrow$) Recimo, da je f največji pretok v omrežju N. Potem po izreku 3.3. velja, da v omrežju N ni več nezasičenih poti.
($\Leftarrow$) Recimo, da ne obstaja nezasičena pot v omrežju N. Naj bo V_s množica vozlišč, ki so iz s dosegljiva po nezasičenih povezavah. Ker ni nezasičene poti od s do t, sledi, da vozlišče $t \notin V_s$. Naj bo $V_t = V_N - V_s$. Potem je $\langle V_s, V_t \rangle$ prerez omrežja N. Zato velja

$$f(e) = \begin{cases} c(e), & \text{če je } e \in \langle V_s, V_t \rangle, \\ 0, & \text{če je } e \in \langle V_t, V_s \rangle. \end{cases}$$

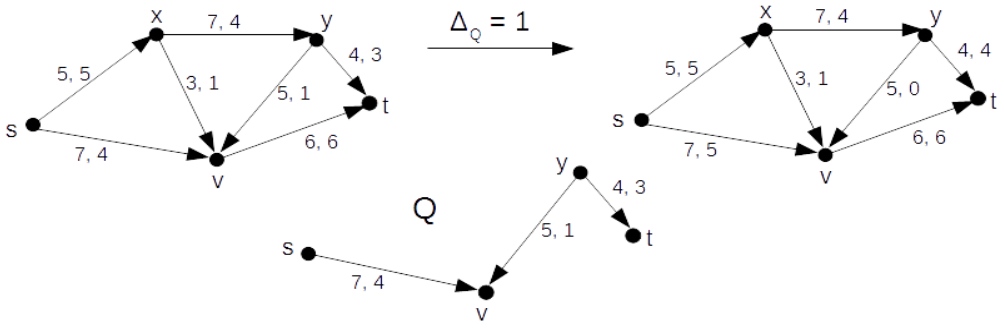
Po izreku 3.2 sledi, da je f največji pretok v omrežju N. □

Na sliki 3.6 imamo omrežje z neničelnim pretokom f. Poiščemo nezasičeno pot, od vozlišča s do vozlišča t. Najdemo nezasičeno pot s, v, y, t. Preko nezasičene poti bomo povečali tok za $\Delta_Q = 1$. Dobili smo nov pretok. Želimo najti novo nezasičeno pot, toda povezavi yt in vt sta že zasičeni in ne moremo priti do vozlišča t. Trenutni pretok f je torej največji pretok in $|f| = 10$.

Posledica 3.1. *Za dano omrežje je vrednost največjega pretoka enaka vrednosti najmanšega prereza.*

Dokaz. Naj bo f največji pretok. Potem konstruiramo prerez $\langle V_s, V_t \rangle$ kot v dokazu izreka 3.4 in ugotovimo, da je $|f| = c\langle V_s, V_t \rangle$. Potem je po izreku 3.2 $\langle V_s, V_t \rangle$ najmanjši prerez. S tem je posledica dokazana. □

Zgornji izreki so osnova za Ford–Fulkersonov algoritem za iskanje največjega pretoka. Opisali ga bomo v naslednjem razdelku.



Slika 3.6: Na nezasičeni poti Q povečamo pretok za $\Delta_Q = 1$

3.2 Ford–Fulkersonov algoritem

V tem razdelku bomo opisali Ford–Fulkersonov algoritem. Algoritem temelji na izreku 3.4. Na vsakem koraku algoritem poišče nezasičeno pot od izvora do ponora in na njej poveča pretok. Postopek ponavljamo in končamo, ko ni več nezasičenih poti. V Ford–Fulkersonovem algoritmu potrebujemo metodo za iskanje nezasičenih poti. Poznamo več metod za iskanje nezasičenih poti. Med te spadata iskanje v širino (breath first search, BFS) in iskanje v globino (depth first search, DFS).

Če imamo opravka s celimi števili, je število korakov končno in enako največ velikosti pretoka, ker se na vsakem koraku pretok poveča vsaj za 1. Podobno velja tudi za racionalna števila. Z algoritmom DFS se lahko zgodi, da imamo eksponentno mnogo korakov, medtem ko z algoritmom BFS je algoritem polinomski. Kadar se uporabljajo iracionalna števila, takrat nimamo zagotovila, da bomo imeli končno število korakov, vendar se v praksi taki primeri skoraj ne pojavljajo. V algoritmu 3 je predstavljen algoritem Ford–Fulkerson. Za iskanje nezasičenih poti uporabimo algoritem 4, ki nezasičeno pot poišče z iskanjem v širino.

Algoritem 3: algoritem Forda in Fulkersona

Input: Omrežje N

Output: Največji pretok v omrežju N

begin

  for za vsako povezavo e v omrežju N **do**
    $f(e) = 0$;
  end
  repeat
    najdi nezasičeno pot Q po algoritmu 4.;
    $\Delta_Q = min_{e \in Q} \{\Delta_e\}$;
    for za vsako povezavo e v Q **do**
      if povezava e je pozitivna **then**
        $f(e) = f(e) + \Delta_Q$;
      end
      else
        $f(e) = f(e) - \Delta_Q$;
      end
    end
  until obstaja nezasičena pot Q v omrežju N;
  vrni pretok f;

end

Izpeljimo časovno zahtevnost za algoritem Forda in Fulkersona. Algoritem najprej uporabi osnovno metodo preiskovanja v širino, ki smo jo opisali v prejšnjem poglavju. Kot smo pokazali, zanj porabimo $O(|V| + |E|)$ časa. Poleg preiskovanja v grafu, moramo skozi graf pripeljati tudi tok f, da bomo lahko določili vrednost F, ki predstavlja največji pretok, ki ga pripeljemo od izvora do ponora. Trenutnemu toku prištejemo vrednost Δ_Q, če je povezava pozitivna, v nasprotnem primeru pa jo odštejemo. Operaciji prištevanja in odštevanja se izvajata v konstantnem času, ker pa moramo pregledati vse povezave, celotna zanka zahteva $O(|E|)$ časa. Ker operacijo pregledovanja nezasičenih poti ponavljamo, dokler ni več nezasičenih poti, dobimo na koncu

največji pretok F. To nam v najslabšem primeru prinese F ponovitev, če vsako iteracijo povečujemo za 1. S seštevanjem zgoraj omenjenih posameznih formul pridemo do končne časovne zahtevnosti $O((|V| + |E|) \cdot F)$. Časovna zahtevnost je lahko eksponentna glede na velikost grafa, če so kapacitete povezav velike.

Algoritem 4: Najdi nezasičeno pot

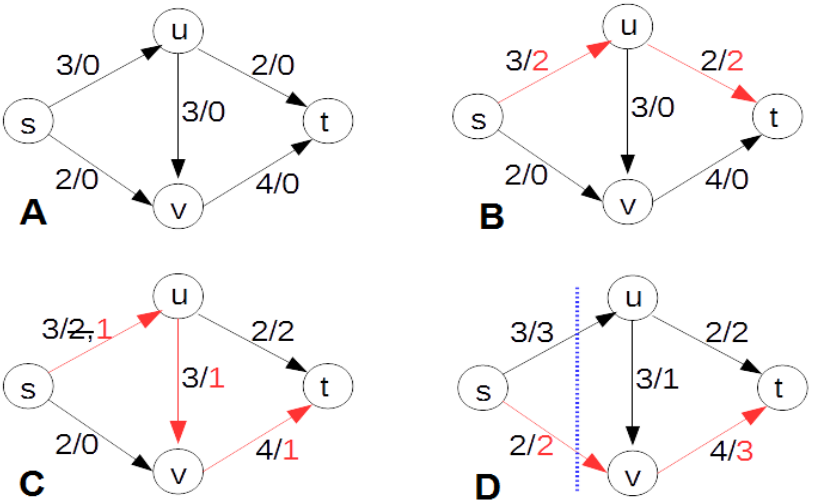
Input: Omrežje N in pretok f v omrežju N

Output: Nezasičena pot Q ali najmanjši prerez s kapaciteto $|f|$

begin
- $V_s = \{s\}$;
- oznaka$[s] = 0$;
- $i = 1$;
- **while** V_s ne vsebuje vozlišča t **do**
  - **if** obstaja nezasičena povezava **then**
    - naj bo e povezava, katere začetek je označeno vozlišče v, ki ima najmanjšo vrednost oznake in katere konec je neoznačen;
    - naj bo w neoznačeno vozlišče povezave e;
    - $nazaj(w) = v$;
    - oznaka$[w] = i$;
    - v množico V_s dodamo vozlišče w;
    - $i = i + 1$;
  - **end**
  - **else**
    - vrni najmanjši prerez $\langle V_s, V_N - V_s \rangle$;
  - **end**
- **end**
- rekonstruiraj pot Q tako, da začnes v točki t in slediš označbam $nazaj(*)$, kjer $*$ pomeni predhodno vozlišče;
- vrni nezasičeno pot Q;

end

Potek Ford–Fulkersonovega algoritma si bomo pogledali na sliki 3.7. Na sliki A je osnovno omrežje, na katerem bomo prikazali potek algoritma. Na povezavah imamo predstavljene vrednosti, kjer prvo število predstavlja kapaciteto povezave, druga vrednost pa trenutni pretok čez povezavo. Na sliki B najprej poiščemo nezasičeno pot in na tej poti povečamo pretok. Pot, ki smo jo našli je s, u, t in je na sliki B označena rdeče. Velikost toka, ki smo ga uspešno pripeljali iz izvora do ponora je 2. Na povezavah na tej poti so se vrednosti iz 0 spremenile v 2 in s tem je povezava (u, t) postala zasičena. Na sliki C smo našli novo nezasičeno pot s, u, v, t. Na povezavi (s, u) lahko pripeljemo samo še 1 enoto toka, ki ga pripeljemo do vozlišča t. Na sliki D smo našli novo nezasičeno pot s, v, t, skozi katero povečamo pretok za 2. Ker ne moremo več najti nezasičenih poti, lahko naredimo najmanjši prerez. Z modro črto smo označili mejo najmanjšega prereza, ki ločuje množici $V_s = \{s\}$ in $V_t = \{u, v, t\}$. Vrednost najmanjšega prereza je enaka velikosti največjega pretoka in je enaka 5.



Slika 3.7: Potek algoritma Ford–Fulkerson.

Leta 1972 sta Jack Edmonds in Richard Karp objavila Edmonds–Karpov algoritem, ki je izboljšana verzija Ford–Fulkerosnovega algoritma [8]. Osnov-

na izboljšava je to, da vedno poiščemo najkrajšo nezasičeno pot.

Izrek 3.5. *Če vzamemo vedno najkrajšo nezasičeno pot, je število korakov v algoritmu Forda in Fulkersona polinomsko glede na velikost omrežja.*

V osnovnem algoritmu Ford–Fulkerson ni določeno, kako poiščemo nezasičeno pot, zato število korakov ni nujno polinomsko. V Edmons–Karpovi izboljšavi algoritma Forda in Fulkersona uporabimo metodo iskanja v širino, ki vedno najde najkrajšo nezasičeno pot. Iz izreka 3.5 sledi, da časovna zahtevnost ni več odvisna od največjega pretoka, temveč samo od iskanja najkrajše poti, ki jo najdemo v času $O(|E|)$. Vsakokrat moramo zasičiti vsaj eno povezavo $O(|E|)$. Razdalja od zasičene povezave do izvora mora biti v vsaki iteraciji daljša od prejšnje, to se zgodi v času $O(|V|)$. Iz tega sledi, da je časovna zahtevnost Edmonds–Karpovega algoritma $O(|V| \cdot |E^2|)$.

3.3 Diničev algoritem

Yefim A. Dinitz je leta 1970 prvič objavil Diničev algoritem [2]. Diničev algoritem ima polinomsko časovno zahtevnost za iskanje največjega pretoka v omrežju. Spada med lažje algoritme za implementacijo in je eden izmed hitrejših algoritmov v praksi.

Algoritem poteka v več fazah. V vsaki fazi vsakemu vozlišču dodelimo nivojsko vrednost, ki predstavlja najkrajšo razdaljo od vozlišča do izvora. Na ta način z metodo preiskovanja v širino konstruiramo nivojski graf. S pomočjo nivojskega grafa lahko pošljemo več različnih nezasičenih poti naenkrat. Zato je Diničev algoritem boljši od Edmonds–Karpovega algoritma. Tako kot v algoritmu Forda in Fulkersona tudi tukaj zasičimo povezave s Δ_Q in nato ponovno izračunamo nivojski graf. Algoritem se konča, ko ne najdemo več nezasičenih poti.

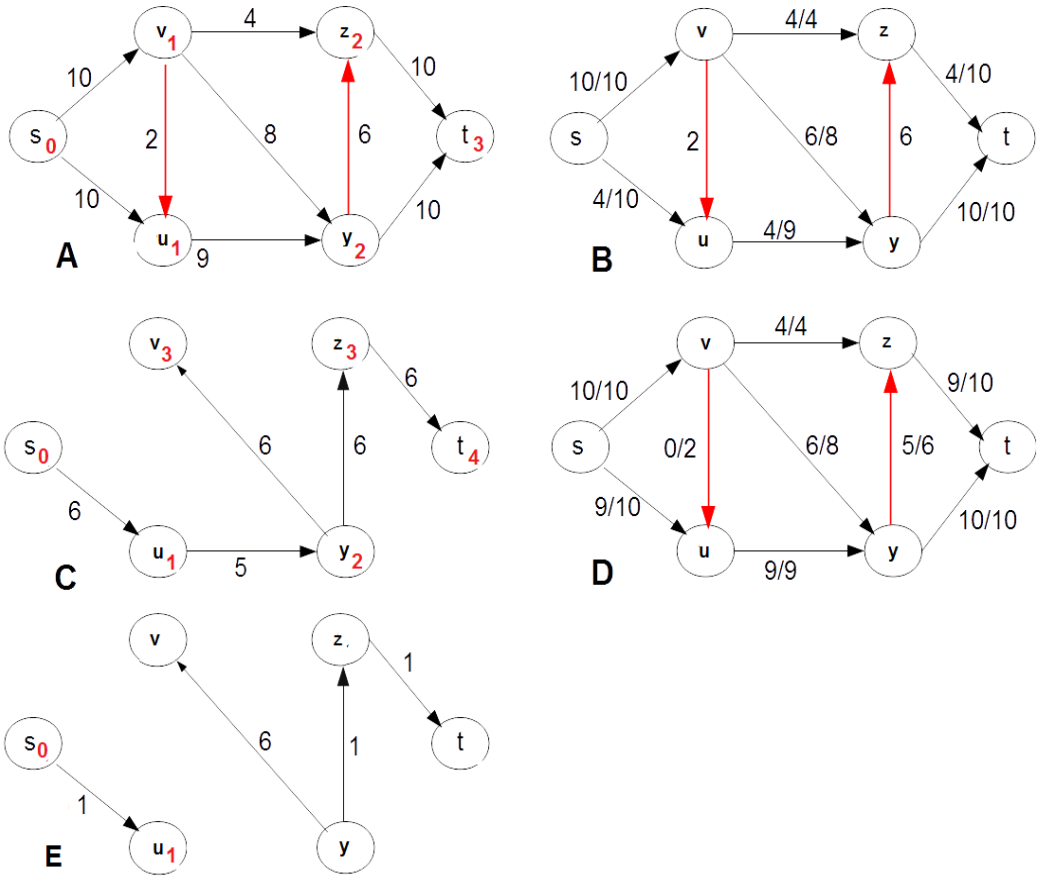
Postopna analiza časovne zahtevnosti Diničevega algoritma 5 nas pripelje do nekaterih izboljšav v času, ki jih zahtevajo posamezne operacije. V grobem lahko algoritem razdelimo na dva večja dela, ki se ponavljata v vsaki iteraciji.

Zunanja zanka se izvede največ $O(|V|)$ krat. Z izboljšanim algoritmom BFS vsako grajenje novega nivojskega grafa zahteva $O(|E|)$ časa. Nato sledijo operacije iskanje poti, povečanje pretoka in brisanja vozlišč. V primeru, da pri povečanju toka po nezasičeni poti naletimo na popolnoma zasičeno povezavo, to brišemo iz nivojskega grafa.

Operacija nam vzame $O(|E|)$ časa. Če pri preiskovanju poti naletimo na osamljeno vozlišče (zaradi prej omenjenega brisanja povezave) tega pobrišemo iz nivojskega grafa, za kar potrebujemo $O(|V|)$ časa. Celoten sklop operacij zahteva $O(|V| \cdot |E|)$ časa. Če seštejemo posamezne časovne zahtevnosti operacij, dobimo $O(|V|) \cdot O((|E|) + O(|E| \cdot |V|))$. Ker upoštevamo le največje prispevke parametrov, dobimo $O(|V|) + O(|E| \cdot |V|)$ in na koncu dobimo časovno zahtevnost $O(|E| \cdot |V|^2)$, kar nam predstavlja tudi časovno zahtevnost celotnega algoritma.

Postopek Diničevega algoritma je predstavljen na sliki 3.8. Na sliki A imamo začetno omrežje. Z rdečimi številkami so označeni nivoji. Z isto barvo so označene tudi povezave, preko katerih tok ne more teči, saj se nahajamo na istem nivoju. Sedaj moramo najti nezasičene poti tako, da prehajamo po nivojih $(0 - 1 - 2 - 3)$. Prva nezasičena pot, ki smo jo naši je s, v, z, t. Vrednost kapacitete povezave (v, z) je najmanjša od ostalih vrednosti kapacitet na tej nezasičeni poti, zato ima pretok, ki gre skozi to nezasičeno pot vrednots 4. S pretokom zasičimo povezave na tej nezasičeni poti in poiščemo novo nezasičeno pot. Nova nezasičena pot, ki jo najdemo je s, v, y, t. Na povezavi (s, v) imamo najmanjšo razliko, med kapaciteto povezave in tokom. Pretok, ki gre preko te nezasičene poti ima vrednost 6. Povezave na tej poti zasičimo in poiščemo novo nezasičeno pot. Nova najdena nezasičena pot je s, u, y, t. Preko te poti lahko pripeljemo 4 enote toka, saj imamo na povezavi (y, t) najmanjšo razliko med kapaciteto povezave in tokom. Zasičimo povezave na tej poti in ponovno poiščemo nezasičeno pot. Ker ne uspemo več najti nezasičene poti, moramo vozliščem na novo opredeliti nivoje. To naredimo, tako da v omrežju odstranimo vse povezave, ki so zasičene in nato vsem vozliščem na novo opredelimo nivoje. Na sliki C imamo predstavljeno

omrežje z novimi dodeljenimi nivoji. Preverimo ali v novem omrežju obstaja nezasičena pot. Najdemo pot s, u, y, z, t, ki ima na povezavi (u, y) najmanjšo razliko med kapaciteto in tokom. Pretok, ki gre preko te nezasičene poti ima vrednost 5. Zasičimo povezave in poiščemo novo nezasičeno pot. Ker ne najdemo nobene nezasičene poti, vozliščem dodelimo nove nivoje. Omrežje z novimi dodeljenimi nivoji, je predstavljeno na sliki E. Ker tudi na tem omrežju ne najdemo nobene nezasičene poti smo prišli do konca algoritma. Našli smo največji pretok, ki ima velikost 19.



Slika 3.8: Postopek Diničevega algoritma.

Algoritem 5: Diničev algoritem

Input: Omrežje $N = ((V, E), c, s, t)$

Output: Največji pretok v omrežju N

```
begin
    Ustvarimo novo podatkovno vrsto Q;
    tok = 0;
    while true do
        obiskan[ ]=-1; //vsako vozlišče, ki še ni bilo obiskano ima
         vrednost -1
        V vrsto Q dodaj vozlišče s;
        obiskan[s] = −2; //označimo izvor kot obiskan
        while Q ni prazen in obiskan[t] == −1 do
            u = Iz Q odstrani prvo vozlišče;
            for za vsako vozlišče v, ki ima povezavo z u do
                //e predstavlja povezavo med vozliščema v in u
                if v ni obiskan in Δ_e > 0 then
                    V vrsto Q dodaj vozlišče v;
                    obiskan[v]=u;
                end
            end
        end
        if obiskan[t] == −1 then
            break;
        end
        for za vsako vozlišče z do
            if m[z][t] > 0 in obiskan[z] ! = −1 then
                ozina = Δ_e; //e je povezava med t in z
                if ozina > 0 then
                    osveži kapacitete na poti od ponora do vozlišča z in
                     naprej do ponora v nivojskem Grafu;
                end
                tok += ozina;
            end
        end
    end
    vrni tok;
end
```

Poglavje 4

Segmentacija slik

V tem poglavju se bomo seznanili s pojmom segmentacije slik. Opisali bomo tudi nekaj metod za segmentacijo slik. Pri pisanju tega poglavja smo si pomagali z viri [13, 9, 6].

Z vidika računalniškega vida je segmentacija slik postopek delitve digitalne slike na več segmentov. Segmentacija slike je proces, ki dodeli vsaki slikovni piki oznako, tako da slikovne pike z isto oznako tvorijo isti segment. Rezultat je množica segmentov, ki skupaj prekrivajo celotno sliko. Cilj segmentacije je poenostavitev slike za lažje analiziranje. Običajno se segmentacija slik uporablja za iskanje predmetov in mej (linije, krivulje,..) na slikah. Pri večjih problemih (prepoznavanje, indeksiranje slike,...) se lahko uporabijo rezultati segmentacije za ujemanje in reševanje problemov ločevanja in prepoznavanja slike po delih. Za razvoj dobre metode želimo, da ta zajame zaznavno pomembne skupine/regije, ki odražajo globalni vidik slike in za določeno število slikovnih pik porabijo skoraj linearni čas.

Segmentacijske tehnike, ki lahko hkrati obdelujejo več slik v sekundi, se uporabljajo v aplikacijah za obdelavo videa. Pri teh metodah segmentacije želimo čim manjšo časovno zahtevnost.

Segmentacijske metode delimo v dve skupini. Prve, ki temeljijo na gručah, in druge, ki temeljijo na grafih. Naštejmo nekaj najbolj znanih metod:

- metoda Otsu (*otsu's method*),
- prelomna segmentacija (*watershed segmentation*),
- razvrščanje v skupine (*clustering methods*),
- pragovna segmentacija (*tresholding method*),
- zaznavanje robov (*edge detection*),
- metoda rasti regij (*region-growing methods*),
- metode, ki temeljijo na parcialnih diferencialnih enačbah (*partial differential equation-based methods*).

Nekatere izmed zgoraj naštetih metod bomo opisali bolj podrobno in predstavili s slikovnim gradivom. Originalno sliko vrtnice sem slikala sama. Sliki hrušk in fotografa sta testni sliki v programu MATLAB. Za prikaz delovanja segmentacijskih metod sem uporabila že implementirane algoritme v programu MATLAB [14].

4.1 Pragovna metoda

V večini primerov poskušamo združiti segmente s podobnimi lastnostmi v iste regije. Najenostavnejši primer je, ko slikovne pike dodeljujemo v regije glede na njihovo intenzivnost. Torej je naravni način segmentacije takih območji, da jih razvrstimo v svetlo ali temno regijo. Pragovna metoda se v glavnem uporablja za ustvarjanje binarnih slik.

Pragovna metoda ustvarja binarne slike iz slik tako, da vsem slikovnim pikam pod določeno mejo dodeli vrednost nič, vsem ostalim pikam pa vrednost ena. Naj $f(x, y)$ predstavlja trenutno vrednost pike (x, y) in naj $g(x, y)$ predstavlja novo vrednost te pike. Potem za nek vnaprej izbrani T definiramo

$$g(x,y) = \begin{cases} 1, & \text{če je } f(x,y) \geq T, \\ 0, & \text{sicer.} \end{cases}$$

Največji problem te metode je, da upošteva zgolj intenzivnost barve, ne pa tudi drugih relacij med slikovnimi pikami. Pogoste težave pri pragovni metodi so odvečne slikovne pike (slika 4.1, območje A) in izgubljene regije v območju B, kjer smo izgubili regijo lista. Poleg omenjenih težav, sence in prekrivanje objektov predstavljajo še zahtevnejši problem za dobro segmentacijo slik.

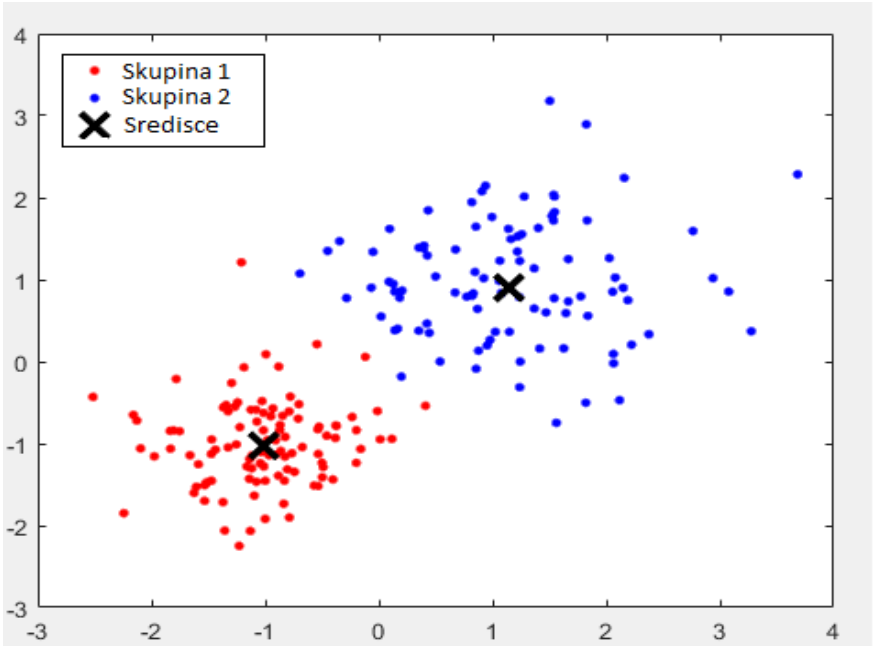
Slika 4.1: Levo je prikazana originalna slika in desno slika, ki jo dobimo s pragovno metodo.

4.2 Razvrščanje v skupine (CM)

Razvrščanje v skupine je proces delitve niza podatkov v podskupine oziroma gruče. Za primer si poglejmo točke v ravnini na sliki 4.2. Skupine bomo razvrstili v podskupine točk, ki so si blizu v evklidskem prostoru [19].

Med CM spadajo metode:

- metoda razvrščanja v skupine z optimalno kriterijsko funkcijo (*evolving clustering method with constrained minimization*),
- algoritem k-tih voditeljev (*k-means clustering*, KMC),
- metoda histogramov (*histogram-based methods*).



Slika 4.2: Set naključno generiranih točk v evklidskem prostoru, ki so razdeljene v dve skupini. Vsaka skupina vsebuje točke, ki so si blizu.

To so algoritmi, ki dobro delujejo z izoliranimi, kompaktnimi gručami ter so intuitivne in pogosto uporabljene metode. Osnovna ideja je najti hierarhično razvrstitev, s katero bomo minimizirali napako glede na nek kriterij med slikovnimi pikami. Najbolj znan kriterij je vsota kvadratov napak, ki meri celotno evklidsko razdaljo med slikovnimi pikami. Napako lahko globalno optimiziramo s pomočjo hevristike.

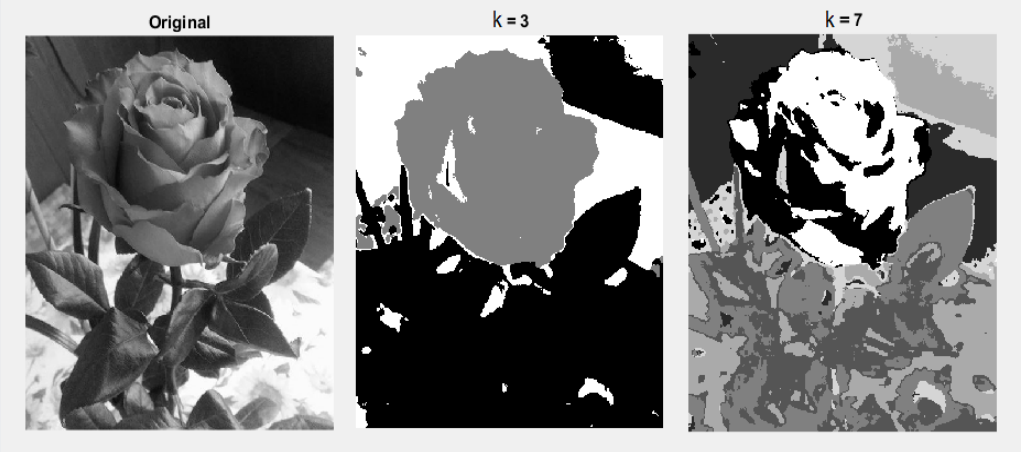
Najenostavnejši in najpogosteje uporabljen je algoritem k-tih voditeljev. Algoritem se uporablja za minimiziranje vsote kvadratov napak v evklidskem prostoru. Ta algoritem podatke spravi v K skupin $(C_1, C_2, ..., C_K)$, ki so predstavljene z njihovimi centri. Središče vsake skupine je izračunano kot povprečje vseh vrednosti, ki pripradajo tej skupini.

KMC se začne z inicializacijo začetnih centrov, ki so lahko izbrani naključno ali hevristično. V vsaki iteraciji je vsaka slikovna pika dodeljena najbližjemu centru skupine glede na evklidsko razdaljo med njima. Cilj je

minimizirati napako.

$$D = \sum_{j=1}^{k} \sum_{i=1}^{n} \parallel x_i^j - c_j \parallel^2.$$

Pri tem je k število skupin, n število slikovnih pik, x_i trenutna slikovna pika in c_j središčna vrednost skupine j. Algoritem ima linearno časovno zahtevnost in je le eden izmed razlogov, zakaj je algoritem zelo priljubljen in uporabljen.



Slika 4.3: Algoritem k-tih voditeljev: prva slika je originalna, za drugo sliko izberemo $k = 3$ (imamo 3 barve) in za zadnjo sliko izberemo $k = 7$.

Metodo k-tih voditeljev se združuje z drugimi algoritmi z namenom boljših rezultatov same segmentacije. Predvsem se metoda k-tih voditeljev uporablja v bioinformatiki [20].

4.3 Metoda regij

Med preprostejše metode segmentacije slik spada metoda regij. Regije na sliki so skupine med seboj povezanih slikovnih pik s podobnimi lastnostmi. Metoda regij sliko razdeli v različne regije z vnaprej določenimi kriteriji kot so barve, intenzivnost ali predmeti. Segmentacija slik z metodo regij uporablja tri glavne metode:

- metoda rasti regij (*region growing*),
- metoda delitve regij (*region splitting*),
- metoda združevanja regij (*region merging*).

Metoda rasti regij je proces, pri katerem odstranjujemo regije iz slike po vnaprej določenih merilih. Povečevanje regije je pristop k segmentaciji, pri katerem sosedne slikovne pike združimo v regijo brez robov. Ta postopek se iterativno ponavlja. Če najdemo sosednjo regijo, z uporabo metode združevanja regij, šibke robove izbrišemo in močne robove obdržimo. Metoda delitve regij celotno sliko razdeli na več podregij in ta postopek ponavlja dokler je to mogoče. Metoda združevanja regij združuje podregije v večje regije, ki imajo podobne lastnosti. Postopek končamo, ko ne moremo več združiti nobene regije. Metoda delitve in združevanja regij se običajno izvaja s podatkovnimi drevesi.

Metodo regij največkrat uporabimo, ko želimo ločiti dve med seboj podobni regiji. Primer je slika 4.4, pri kateri smo metodi podali vrednosti x in y, s katerima določimo začetek nove regije. V programu imamo že določeno vrednost t, ki nam pove, kako veliko območje bomo pregledovali. Območje, ki ga pregledujemo so točke, ki se od (x, y) v prvi in drugi koordinati razlikujejo za največ t. Slikovne pike, ki se nahajajo na tem območju in je razlika med njimi in slikovno piko xy majhna, se dodajo v regijo. Dobra lastnost metode regij je tudi enostaven koncept algoritma, vendar je prostorsko zelo potratna.

Slika 4.4: Na sliki levo je originalna slika hrušk in na sliki desno segmentacija slike z metodo rasti regij.

V medicini se za segmentacijo slik pogosto uporablja metoda rasti regij v kombinaciji z algoritmom watershed, zaradi potrebe dobre segmentacije mikroskopiranih slik (majhne strukture) [18].

4.4 Segmentacije na grafih s pretokom

Segmentacija na grafih z uporabo največjega pretoka je glavna tema te diplomske naloge in bo bolj natančno predstavljena v naslednjem poglavju. Tu jo bomo le na kratko opisali.

Leta 1989 so prvič ugotovili, kako dobra je metoda najmajšega prereza za probleme v računalniškem vidu. Pokazali so namreč, da je najmanjši prerez uporaben pri restavriranju binarnih slik. Na žalost je ta tehnika ostala neopažena skoraj 10 let. V poznih 90-tih so z novimi tehnikami računalniškega vida pokazali, kako se lahko uporablja najmanjši prerez v nebinarnih slikah. Kasneje so jo začeli uporabljati tudi v n-dimenzionalnih slikah, pri katerih z njeno pomočjo pridobimo željeni objekt ali pa po robovih obrisan objekt [4].

Glavna značilnost segmentacije na grafih je, da prevedemo segmentacijo slike na problem največjega pretoka. Naj bo $G = (V, E)$ neusmerjen graf. Slikovne pike predstavljajo vozlišča v grafu G, dodatno imamo še vozlišči s in t, ki predstavljata izvor oziroma ponor pri problemu največjega pretoka. Definirano imamo vrednost $T \geq 0$, s katero določimo ali vozlišče pripada ospredju ali ozadju slike. Če ima vozlišče večjo vrednost od T, potem pripada ospredju in je povezano z vozliščem s, drugače pripada ozadju in je povezano z vozliščem t. Povezave, katerih začetno ali končno vozlišče je izvor oziroma ponor, imajo uteži določene z vrednostjo slikovne pike. Ostalim povezavam pripadajo nenegativne uteži, ki predstavljajo razliko med vrednostima sosednjih vozlišč. Segmentacijo pretvorimo v problem najmanjšega prereza. Želimo najti tak prerez, ki najboljše ločuje vozlišča z večjimi vrednostmi od tistih z manjšimi vrednostmi. Primer segmentacije na grafih je predstavljen na sliki 4.5.

Slika 4.5: Levo je slika fotografa, desno pa je slika po segmentaciji, na kateri je izbrani objekt fotograf.

Poglavje 5

Segmentacija s pretokom

Eden od problemov pri obdelavi slik je segmentacija slike v različno usklajene regije. Kot primer lahko vzamemo sliko, na kateri trije ljudje stojijo pred kompleksnim ozadjem. Zaradi ozadja je težko identificirati vsako osebo posebej kot samostojen objekt. Eden izmed osnovnih problemov je razvrstitev slikovne pike ospredju ali ozadju. Izkaže se, da se to odlično reši z najmanjšim prerezom na ustreznem grafu. Pri pisanju tega poglavja smo si pomagali z viroma [11, 15].

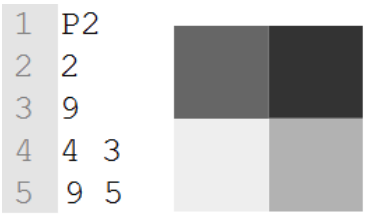
Slika je orodje, s katero prikazujemo vizualno percepcijo. Slika je lahko dvodimenzionalna, kot na primer fotografija, ali tridimenzionalna, kot so kipi in hologrami. Slika, kot jo bomo obravnavali v tem poglavju, je funkcija dveh spremenljivk $f(x, y)$, pri kateri sta x in y koordinati slikovne pike v ravnini. Z vrednostjo $f(x, y)$ je predstavljena svetlost točke (x, y).

Dano imamo sivinsko sliko, ki je predstavljena v formatu PGM (Portable Graymap Format). V formatu PGM je slika predstavljena s posebnimi začetnimi vrsticami, ki se nadaljujejo z vrednostmi slikovnih pik:

- vrsta formata (P2),
- dolžina in širina slike v pikah,
- največja vrednost slikovne pike, ki je uporabljena na sliki,

- sledijo vrstice vrednosti slikovnih pik, ki predstavljajo sliko.

Zgled slike, ki je predstavljena v formatu PGM, je na sliki 5.1.

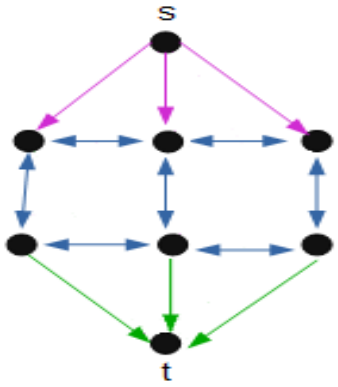


Slika 5.1: Opis slike s formatom PGM in desno primer slike.

Sliki priredimo usmerjen graf $G = (V, E)$ z vozlišči $u_i \in V$ in povezavami $(u_i, u_j) \in E$. Slikovne pike predstavljajo vozlišča množice V, povezave predstavljajo množico vseh parov sosednjih slikovnih pik. Vsako vozlišče u_i je predstavljeno z nenegativno vrednostjo slikovne pike, označimo jo s $f(u_i)$. Vozlišča želimo razdeliti med množici A in B. V množici A so tista vozlišča, ki pripadajo ospredju, ostala vozlišča pripadajo množici B. Da bomo vozlišča lahko delili med množici, si izberemo poljubno vrednost parametra $T \geq 0$. Če za označeno vozlišče velja $f(u_i) > T$, potem vozlišče dodamo v množico A, sicer vozlišče pripada množici B. Odločitve, ki jih sprejmemo za sosede vozlišča u_i, veljajo posledično tudi za vozlišče u_i. Kadar ima vozlišče u_i večino sosednjih vozlišč dodeljenih ozadju, lahko iz tega sklepamo, da je tudi vozlišče u_i v ozadju. S tem postane označevanje vozlišč enostavnejše.

Sedaj imamo vozlišča razdeljena v množici A in B, ki predstavljata ospredje in ozadje. Naš cilj je najti najmanjši prerez. Ustvarimo novo vozlišče s, ki predstavlja ospredje in vozlišče t, ki predstavlja ozadje. Vozlišča, ki pripadajo množici A bomo povezali z vozliščem s, ostala vozlišča, ki pripadajo množici B bomo povezali z vozliščem t (slika 5.2), določimo še $f(s) = 0$ in $f(t) = 0$.

Definiramo torej nov graf $G' = (V', E')$, pri katerem množica V' sestoji iz množice V in dodatnima vozliščema s in t. Množica E' sestoji iz množice E in dodanimi povezavami od s do vsakega vozlišča iz A in od vsakega vozlišča iz B do t. Za izračun najmanjšega prereza moramo imeti definirane vrednosti



Slika 5.2: Vozlišča, ki ustrezajo sosednjim pikam na sliki, so povezana. Vozlišča, ki pripadajo ospredju so povezana z vozliščem s, ostala vozlišča pa z vozliščem t.

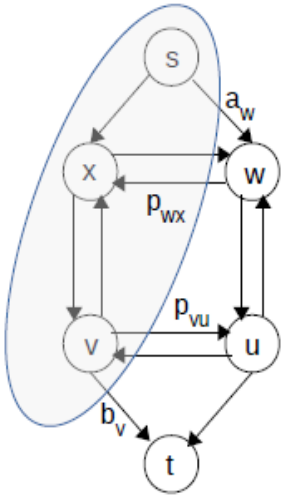
na povezavah. Naj bo $P : E \to \mathbb{R}^+$ preslikava, ki vsaki usmerjeni povezavi v množici E priredi vrednost $p_{ij} =\mid f(u_i) - f(u_j) \mid$. Za vsako povezavo (s, j) velja, da je njena kapaciteta povezave enaka $a_j = f(u_j)$. Povezava (i, t) ima kapaciteto povezave enako $b_i = f(u_i)$. Če gre prerez skozi povezavo (s, j), ta h kapaciteti prereza prispeva vrednost a_j. Če prerez seka povezavo (i, t), ta h kapaciteti prereza prispeva vrednost b_i in če gre prerez skozi povezavo (i, j), je prispevana vrednost p_{ij}. Z naslednjo formulo izračunamo kapaciteto prereza

$$c\langle V'_s, V'_t\rangle = \sum_{i\in A} a_i + \sum_{j\in B} b_j + \sum_{\substack{i\in A,\\ j\in B}} p_{ij}.$$

Sedaj moramo najti le najmanjši prerez, za kar lahko uporabimo naprimer algoritem Forda in Fulkersona.

Na sliki 5.3 vidimo prerez omrežja. Povezava (s, w) prispeva h kapaciteti prereza vrednost a_w. Povezavi (w, x) in (v, u) prispevata h kapaciteti prereza vrednosti p_{wx} in p_{vu}, medtem ko povezava (v, t) prispeva vrednost b_v.

V nadaljevanju bomo predstavili algoritme za opisano segmentacijo na grafih. V algoritmu 6 je opisan postopek segmentacije s pretokom. Znotraj



Slika 5.3: Označene povezave prispevajo vrednosti h kapaciteti najmanjšega prereza.

algoritma kličemo različne metode, s katerimi ustvarimo omrežje, na katerem kasneje poiščemo največji pretok. Prvi korak v algoritmu je, da preberemo vrednosti iz datotek PGM. Naj bo m število vrstic in n število stolpcev. Prebrati moramo vse vrednosti slikovnih pik, zato je časovna zahtevnost tega algoritma enaka $O(m \cdot n)$. Zapomnimo si dolžino, širino in največjo vrednost slikovne pike. Vrednosti slikovnih pik zapišemo v matriko P. Iz zgleda 5.1 imamo matriko

$$P = \begin{bmatrix} 4 & 3 \\ 9 & 5 \end{bmatrix}.$$

Inicializiramo novo prazno matriko M z dodatnimi stolpci in vrsticami za predstavitev vrednosti s in t. Pokliče se metoda za kreiranje matrike sosednosti, ki predstavlja matriko z vrednostmi povezav med sosednjimi vozlišči. Algoritem 7 kot vhodne podatke sprejme, matriko M, matriko P z vrednostimi slikovnih pik, dolžino d in največjo vrednost slikovne pike.

Začetna vrstica in stolpec ter zadnja vrstica in stolpec matrike M so

rezervirani za vrednosti vozlišča s in t. Za vsako slikovno piko izračunamo uteži na povezavah med sosednjimi vozlišči. Uteži predstavljajo razliko med vrednostmi slikovnih pik in jih zapišemo v matriko M:

$$M = \begin{array}{c} \\ s \\ \\ \\ \\ \\ t \end{array} \begin{array}{c} \begin{array}{cccccc} s & & & & & t \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 5 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 5 & 0 & 0 & 4 & 0 \\ 0 & 0 & 2 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Algoritem 6: Postopek segmentacije s pretokom

Input: slika v formatu PGM

Output: binarna slika v formatu PGM slika

begin

Preberemo datoteko PGM v matriko P;

Inicializiramo vrednosti dolžina, širina in največja vrednost;

int[][] M = new int[dolžina * širina +2][dolžina * širina + 2];

Ms = matrikaSosednosti(M, P, dolžina, širina, največja vrednost);

F = IskanjeNajvečjegaPretoka(Ms);

izpis nove datoteke PGM.

end

V algoritmu 7 je zapisan postopek zapisa v matriko M. Ali neka povezava obstaja, lahko v matriki sosednosti preverimo v času $O(1)$. Vendar posledično porabimo $O(|V|^2)$ pomnilnika. Če želimo pregled vseh predhodnikov ali naslednikov izbranega vozlišča, porabimo vedno $O(|V|)$ časa. Druga slabost, ki jo tudi opazimo pri matriki sosednosti, je, da porabimo $O(|V|^2)$ časa, za pregled vseh povezav. Zato je bolj smiselno za predstavitev grafa uporabiti povezan seznam (linked list). Njegove prednosti so, da porabimo le $O(|V| + |E|)$ prostora in $O(|V| + |E|)$ korakov za pregled vseh povezav.

V času $O(|V|)$ lahko preverimo, ali povezava obstaja, ter dodamo ali odstranimo povezavo.

Algoritem 7: Matrika sosednosti

```
Input: matrika M, matrika P, dolžina d, največja vrednost
Output: matrika sosednosti
begin
    vozlisceId = 1;
    vsota = 0 ;                                   // vsota vozlišč
    for int i = 0; i < P.length; i++ do
        for int j=0; j < P[0].length; j++ do
            vsota+ = M[i][j];
            if nismo na koncu vrstice then
                M[vozlisceId][vozisceId + 1] =| (P[i][j] − P[i][j + 1]) |;
            end
            if nismo na začetku vrstice then
                M[vozlisceId][vozisceId − 1] =| (P[i][j] − P[i][j − 1]) |;
            end
            if nismo v zadnji vrstici then
                M[vozlisceId][vozisceId + d] =| (P[i][j] − P[i + 1][j]) |;
            end
            if nismo v prvi vrstici then
                M[vozlisceId][vozisceId − d] =| (P[i][j] − P[i − 1][j]) |;
            end
        end
        vozlisceId + +;
    end
    povprecnaVrednost = vsota/vozlisceId;
    Ms = matrikaSosednostiZIzvoromInPonorom(M, P,
     povprecnaVrednost, največja vrednost);
    vrni Ms;
end
```

V algoritmu 7 moramo vsako vozlišče iz A povezati z vozliščem s in vsako

vozlišče iz B moramo povezati s t, zato pokličemo metodo matrikaSosednostiZIzvoromInPonorom, ki je zapisana v algoritmu 8. S to metodo v vsak začetek stolpca zapišemo vrednost s in na vsakem koncu vrstice zapišemo vrednost t. Časovna zahtevnost, ki jo doprinese ta metoda je $O(2 \cdot |V|)$, samo sprehodimo se čez kvadratno matriko in vsako vrednost povežemo z izvorom in ponorom.

Po algoritmu 8 dobimo novo matriko

$$Ms = \begin{bmatrix} 0 & 0 & 0 & 9 & 5 & 0 \\ 0 & 0 & 1 & 5 & 0 & 4 \\ 0 & 1 & 0 & 0 & 2 & 3 \\ 0 & 5 & 0 & 0 & 4 & 0 \\ 0 & 0 & 2 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

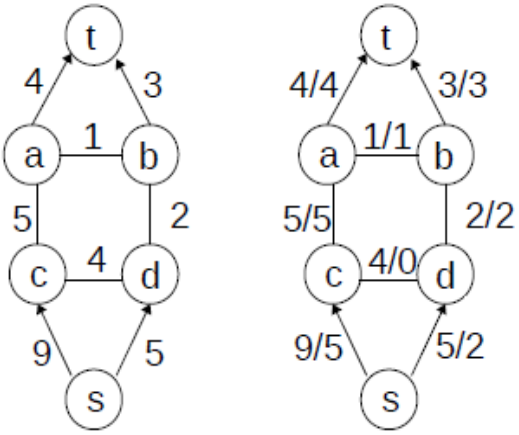
Algoritem 8: Matrika sosednosti z izvorom in ponorom

Input: matrika sosednosti M, matrika P, povprečna vrednost avg

Output: matrika sosednosti z izvorom in ponorom

```
begin
    vozlisceId=1;
    for int i = 0; i < P.length; i++ do
        for int j = 0; j < P[0].length; j++ do
            if P[i][j] >= avg then
                M[1][vozlisceId] = P[i][j]
            end
            else
                M[vozlisceId][M.length-1] = P[i][j];
            end
        end
        vozlisceId++;
    end
    return M;
end
```

V prvi vrstici matrike imamo zapisane vrednosti povezav, katerih vozlišča pripadajo ospredju. Če vozlišče pripada ozadju, je vrednost enaka 0. V zadnjem stolpcu matrike so zapisane vrednosti povezav, katerih vozlišča pripadajo ozadju. Če vozlišče priprada ospredju, je vrednost enaka 0. Matriko Ms uporabimo za izračun največjega pretoka. Na sliki 5.4 levo imamo predstavljno naše omrežje. Povezave, ki imajo kapaciteto 0, na sliki niso označene, saj ne prispevajo ničesar h največjemu pretoku. Vozlišča, ki ustrezajo slikovnim pikam z vrednostmi $4, 3, 9, 5$ smo označili z a, b, c, d. Na sliki 5.4 desno je predstavljeno omrežje, skupaj z največjim pretokom, ki ima velikost 7. Najmanjši prerez $\langle V'_s, V'_t \rangle$ predstavljata množici $V'_s = (s, c, d)$ in $V'_t = (t, a, b)$.



Slika 5.4: Levo začetno omrežje in desno omrežje z danim največjim pretokom.

Primer dobljene segmentacijske slike je prikazan na sliki 5.5. V našem primeru smo se odločili, da bomo vozlišča, ki pripadajo ospredju posvetlili, ostala pa potemnili.

Metode in algoritmi, ki smo jih opisali v tem poglavju, so primerni predvsem za manjše probleme, saj nam pri večjih zmanjka prostora, oziroma je časovna zahtevnost že tako velika, da raje posežemo po drugih algoritmih.

Slika 5.5: Levo prikaz originalne slike in desno je slika po segmentaciji.

Poglavje 6

Sklepne ugotovitve

Cilj diplomske naloge je predstavitev segmentacije slik na grafih s pomočjo največjega pretoka. Podrobno smo si ogledali problem največjega pretoka in najmanjšega prereza ter povezavo med njima. Opisali smo algoritem Forda in Fulkersona in Diničev algoritem, katera uporabljamo za iskanje največjega pretoka.

V nadaljevanju smo predstavili segmentacijo slik. To je postopek, pri katerem s pomočjo algoritmov naredimo novo sliko, ki je lažja za razumevanje in analizo. Običajno se na sliki osredotočimo na objekt, ki ga želimo izločiti iz slike. Predstavili smo različne metode za segmentacijo slik med drugim pragovno metodo, metodo k-tih voditeljev, metodo regij in segmentacijo na grafih. V zadnjem poglavju smo predstavili reševanje problema segmentacije slik na grafih. Izpeljali smo tudi algoritme za reševanje tega problema. Za lažje razumevanje smo opisali tudi primer in ga predstavili na sliki.

Zaradi hitrega napredka tehnologije se lahko pojavijo prostorski problemi hranjenja podatkov, kot tudi kompleksnejše zahteve strank. Zato imamo pri tovrstni segmentaciji različne probleme.

Prvi izmed problemov v segmentaciji na grafih je določanje fiksnega parametra T. V našem primeru smo določili le eno fiksno vrednost, po kateri smo delili slikovne pike v množici A in B. Lahko bi jih določili tudi več, vendar bi zaradi tega morali razdeliti sliko na več delov, ter posameznemu

delu dodeliti eno vrednost, za kar pa bi porabili veliko časa. Pri algoritmih, ki jim določimo več parametrov, preostanek algoritma ostaja enak, glej [21].

Drugi problem segmentacije na grafih je določanje tankih objektov. Čeprav je segmentacija na grafih učinkovita metoda za iskanje objektov, pri objektih, ki so zelo podobnih barv kot ozadje ali pa so zelo tanki in jih zaradi tega težko ločimo od ozadja, metoda odpove. Primer, pri katerem ne moremo uporabiti te metode, je segmentacija slik krvnih žil [5].

Zadnji problem, ki ga bomo navedli je časovna zahtevnost. V današnjem času se resolucije slik hitro izboljšujejo. Posledica tega je, da imajo slike veliko več slikovnih pik. Na primer v biomedicini ima lahko ena slika več milijonov slikovnih pik. Problem je, da segmentacije slike ne moremo prevesti na problem grafa, ker je časovno zelo zahtevno. Zaradi tega so metode kot so segmentacija na grafih primorane k spremembam definiranja grafov. Eden izmed načinov izboljšave je uporaba hevristike v povezavi z drugimi algoritmi, kot je na primer algoritem watershed [12].

Kljub pomanjkljivostim, ki smo jih našteli, je postopek segmentacije z uporabo pretoka zelo uporaben v določenih primerih, naprimer pri računalniški tomografiji (CT) [10].

V diplomi smo predstavili tudi druge metode za segmentacijo. Vendar ne obstaja popolna metoda, ki bi zagotovila najboljšo segmentacijo slike, saj na njo vplivajo različni dejavniki, kot so vrednosti slikovnih pik, podobnosti objektov na slikah in velikost slike. Zato se za segmentacijo slik pogosto uporablja različna kombinacija segmentacijskih metod.

Literatura

[1] Ravindra K Ahuja and James B Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

[2] Yefim Dinitz. Dinitz' algorithm: The original version and even's version. In Oded Goldreich, Arnold L. Rosenberg, Alan L. Selman, editor, *Theoretical Computer Science,* 218–240. Springer-Verlag Berlin Heidelberg, 2006.

[3] Gašper Fijavž. *Diskretne strukture*. Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Ljubljana, 2015.

[4] Yuri Boykov, Gareth Funka-Lea. Graph cuts and efficient N-D image segmentation. *International journal of computer vision*, 70(2):109–131, 2006.

[5] Ali Kemal Sinop, Leo Grady. A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on,* 1–8. IEEE, 2007.

[6] Pedro F. Felzenszwalb, Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Internation Journal of Computer Vision*, 59(2):167–181, 2014.

[7] L. R. Ford, Jr and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[8] Jack Edmonds, Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Association for Computing Machinery*, 19(2):248–264, 1972.

[9] Pratibha Goyal, Manjot Kaur. A review on region based segmentation. *International Journal of Science and Research (IJSR)*, 4(4):3194–3197, 2015.

[10] Chen Yu ke, Wu Xiaoming, Cain Ken, Ou Shan-xin. CT image segmentation based on clustering and graph-cuts. *Procedia Engineering*, 15:5179–5184, 2011.

[11] Eva Tardos, John Kleinberg. *Algorithm Design.* Pearson/Addison-Wesley, Boston, 2006.

[12] Nicolas Lermé, François Malgouyres, Lucas Létocart. Reducing graphs in graph cut segmentation. In *Image Processing (ICIP), 2010 17th IEEE International Conference on,* 3045–3048. IEEE, 2010.

[13] Lior Rokach, Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[14] Matlab optimization toolbox, 2015a. The MathWorks, Natick, MA, USA.

[15] Maria Petrou, Costas Petrou. *Image Processing: The Fundamentals.* Chichester, Wiley, 2010.

[16] Binary Image Segmentation Using Graph Cuts. Dosegljivo na: `http://people.csail.mit.edu/yingyin/resources/doc/projects/yingyin_6854_project.pdf`. [Dostopano: 20. 4. 2017].

[17] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

[18] Padmakant Dhage, Dr. S. K. Shah. Watershed and region growing segmentation tumor detection. *International Journal of Engineering research and Technology*, 4:1061–1067, 2015.

[19] Linda G. Shapiro, George C. Stockman. *Computer Vision*. Prentice Hall, Upper Saddle River, 2001.

[20] Alexander Sturn, John Quackenbush, Zlatko Trajanoski. Genesis: cluster analysis of microarray data. *Bioinformatics*, 18(1):207–208, 2002.

[21] Bo Peng, Olga Veksler. Parameter selection for graph cut based image segmentation. *BMVC*, 32:42–44, 2008.

[22] Robin J. Wilson, John J. Watkins. *Uvod v teorijo grafov*. DMFA, Ljubljana, 1997.

[23] Jonathan Gross, Jay Yellen. *Graph Theory and its Applications*. CRC Press, Boca Raton, 1998.